VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Multimediální platforma pro Java SE a Android

Multimedia Platform for Java SE and Android

2012 Petr Buček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Buček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Multimediální platforma pro Java SE a Android
Multimedia Platform for Java SE and Android

Zásady pro vypracování:

Cílem práce je vytvořit jednoduchou a rychlou platformu pro vývoj aplikací v jazyce Java, které bude možné spustit jak na PC (Java SE), tak na zařízeních se systémem Android. Základem bude objektová nástavba pro OpenGL (vykreslování grafiky a textu), řízení zdrojů a zvuku, která bude mít pro oba systémy jednotné API.

1. Příprava, popis a návrh API, volba vhodných nástrojů, licenční podmínky.
2. Vytvoření základního balíčku pro práci s OpenGL a zdroji.
3. Implementace tříd pro pokročilou práci s grafikou a texturami.
4. Implementace rozhraní pro vykreslování textu.
5. Implementace zvukového rozhraní.
6. Vytvoření několika jednoduchých demonstrativních aplikací.
7. Testování platformy na obou systémech (Android a Java SE), vzájemné srovnání.

Seznam doporučené odborné literatury:

Cay S. Horstmann, Core Java(TM), Volume I--Fundamentals, Prentice Hall, 2007, ISBN-13: 978-0132354769
Reto Meier, Professional Android 2 Application Development, Wrox, 2010, ISBN-13: 978-0470565520
Sayed Hashimi, Pro Android 2, Apress, 2010, ISBN-13: 978-1430226598

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michal Krumnikl**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2012 ..... Petr Bučok

Abstrakt

Cílem práce je vytvořit jednoduchou a rychlou platformu pro vývoj aplikací v jazyce Java, které je možné spustit jak na PC (Java SE), tak na zařízeních se systémem Android. Základem je objektová nástavba pro OpenGL (vykreslování grafiky a textu), řízení zdrojů a zvuku, která má pro oba systémy jednotné API.

Klíčová slova: Java, OpenGL, JSON, Java SE, Android, 2D grafika, multimédia, interaktivita

Abstract

The target of this work is a simple and fast platform for developing applications in Java language which can be run on PC as well as on devices with Android. The base of this work is object extension for OpenGL (graphics and texts rendering), resources and sound controlling API, shared for both systems.

Keywords: Java, OpenGL, JSON, Java SE, Android, 2D graphics, multimedia, interactive

Seznam použitých zkratek a symbolů

ADT	–	Android Development Toolkit
AWT	–	Abstract Window Toolkit
BSD	–	The BSD 3-Clause License (odvozeno od Berkeley Software Distribution)
CSV	–	Comma-Separated Values
DOM	–	Document Object Model
GPL	–	GNU General Public License 3.0
GLU	–	OpenGL Utility Library
GLUT	–	OpenGL Utility Toolkit
HTTP	–	Hyper Text Transfer Protokol
CDL	–	Comma Delimited Lists
IEEE 754	–	Standard IEEE pro dvojkovou aritmetiku v pohyblivé řádové čárce
Java SE	–	Java Standart Edition
JDK	–	Java Development Kit
JSON	–	JavaScript Object Notation
JSR	–	Java Specification Requests, součást Java Community Process
LGPL	–	GNU Lesser General Public License 3.0
LWJGL	–	The Lightweight Java Game Library
MIME	–	Multipurpose Internet Mail Extensions
MIT	–	The MIT License (odvozeno od Massachusetts Institute of Technology)
NEWT	–	JOGL's High Performance Native Windowing Toolkit
OpenAL	–	Open Audio Library
OpenCL	–	Open Computing Language
OpenGL	–	Open Graphics Library
UTF-8	–	UNICODE Text Format
SAX	–	Simple API for XML
SGML	–	Standard Generalized Markup Language
StAX	–	Simple APIs for XML
W3C	–	World Wide Web Consortium
XML	–	eXtensible Markup Language
YAML	–	YAML Ain't Markup Language

Obsah

1 Úvod

Pro vývoj multimediálních aplikací a her lze v dnešní době použít mnoho různých prostředí, platforem, knihoven a enginů. Vývojáři tedy stojí před problémem, které řešení je pro jejich budoucí aplikaci vhodné. Různá řešení umožňují výslednou aplikaci spustit na různých operačních systémech nebo zařízeních. Autoři takových řešení se snaží, aby se dala výsledná aplikace spouštět nezávisle na použitém operačením systému nebo stroji.

S tím souvisí i volba programovacího jazyka. Mezi multiplatformní jazyky, které slouží pro psaní aplikací, patří C++, Java a C#. Současně existuje mnoho nástrojů a knihoven, které s těmito jazyky pracují nebo jsou v nich přímo napsány. Lze také využít moderních skriptovacích jazyků, jako jsou JavaScript nebo LUA. Konkrétně LUA byla přímo vytvořena jako jednoduchý zabudovatelný jazyk, čehož využívá i řada významných her, včetně titulů společnosti Blizzard Entertainment jako World of Warcraft nebo produktu CryENGINE2 od společnosti Crytek. [13]

Podstata mé práce je prozkoumat kompatibilitu desktopové Javy a operačního systému Android. Existuje několik projektů (viz. 1.2), které se zabývají obdobnou tématikou. Mým cílem je navrhnout vlastní platformu, která bude obsahovat vše potřebné pro plnohodnotné psaní aplikací, a zhodnotit použité postupy a mechanismy.

Úvodní kapitola se věnuje shrnutí všech vlastností, které od své budoucí platformy očekávám, a porovnává ji s jinými projekty, které již existují. Dále se věnuje rozdílům mezi desktopovou Javou a systémem Android, popisu na platformě nezávislých technologií, rozdílům v implementaci a dalším prvkům, které je nutné pro vývoj mé platformy vybrat.

Druhá kapitola popisuje samotné API, včetně načítání zdrojů, popisu rozhraní pro aplikace a rozdělení celého projektu na jednotlivé části, které korespondují s Javovskými balíčky v projektu.

Kapitola 3 rozebírá hlavní balíček, který se stará o zavedení samotné platformy a následně i aplikace. Řídí přístup ke zdrojům, souborům i uživatelskému vstupu. Jsou v něm obsaženy třídy pro inicializaci plátna v OpenGL a všechny prvky, které jsou pro chod platformy nutné.

Základní balíček, tzv. jádro, je popsán v kapitole 4. Jádro obsahuje nezávislé třídy pro práci s oblastmi, body, barvami, směry apod., které bylo nutné pro potřeby mé platformy znovu napsat a přizpůsobit.

Datový balíček, popsaný v kapitole 5, dává programátorovi k dispozici nástroje pro práci s různě kódovanými a komprimovanými daty. Jeho úkolem je rovněž čtení souborů se zdroji. Umožňuje práci s kódováním Base64, komprimačními metodami GZIP a DEFLATE nebo zpracování formátu JSON. Rovněž slouží pro práci s čistými obrazovými daty ve formě pixelů.

Grafický a zvukový balíček jsem umístil do kapitoly 6. Je zde popsán způsob vykreslování grafických primitiv a práce se zvukem.

V sedmé kapitole jsou pak popsány ukázkové aplikace, které jsou velmi jednoduché a demonstrují funkčnost platformy. Poslední kapitola shrnuje výsledky celé práce.

1.1 Popis platformy

Ve své práci jsem se zaměřil na tvorbu nástroje, který bude poskytovat programové rozhraní pro akcelerované vykreslování dvojrozměrné grafiky. Celý projekt je napsán v jazyce Java a taktéž aplikace pro něj se píší v Javě. Jednoduché shrnutí všech vlastností tohoto projektu:

- plně funkční na *Android 2.3.3* (API 10) a *JavaSE 6.0*
- jednotné, jednoduché a přehledné API
- využívá kódování textu *UTF-8* a sadu *UNICODE*
- podpora *OpenGL ES* 1.x, práce s vrstvami a nástroji pro 2D vykreslování
- jednotné rozhraní pro přístup ke zdrojům (soubory, data) a uživatelskému vstupu
- podpora formátu *JSON* (vycházející z referenční implementace, balíčku *org.json*)
- jednoduché a jasné konfigurační i datové soubory (založeny na JSON)
- načítání textur ze souborů formátu *PNG* a vrstev dlaždic z editoru Tiled
- obecné třídy s podporou základních operací - body, oblasti, barvy, písma, atd.
- rozšířený systém pro zpracování dat (komprimace *GZIP*, *DEFLATE*; kódování *Base64*)
- pokročilá práce s vykreslováním textu (včetně Unicode znaků)
- podpora *sprite* animací
- podpora přehrávání zvuku a hudby (formát *OGG*; kodek Vorbis)[1]

1.2 Porovnání s jinými projekty

Pro názornost jsem vybral několik projektů, které taktéž umožňují tvořit aplikace pro desktopy i přenosná zařízení s Androidem. Následuje jejich stručný popis a srovnání s mým projektem.

[1] Podpora pro zvuk není v práci plně implementována. Projekt byl původně zamýšlen jako práce pro dva studenty, zvuk zůstal součástí zadání, ale nešlo o primární část mé práce. Zvuk je tedy v projektu zastoupen pouze ve zjednodušené formě (bez použití zmíněných kodeků, které jsou naplánovány do pozdějších rozšíření platformy).

1.2.1 libGDX

Jde o projekt pod licencí Apache License 2.0, obsahující sadu tříd pro práci s vykreslováním 2D a 3D. Umožňuje vytvářet aplikace pro desktopy i zařízení se systémem Android. Rozdíly oproti mému projektu jsou v tom, že libGDX využívá i JNI a pro spuštění aplikace na Androidu je nutné upravovat zdrojový kód. Taktéž nedovede načítat jiné formáty fontů než BMFont. Ostatní funkcionalita je podobná. [21]

Domovská stránka projektu: `http://code.google.com/p/libgdx/`

1.2.2 ORX

Podle popisu na webu jde o lehký a přenosný herní engine s otevřeným zdrojovým kódem a podporou pluginů, určený pro vývoj 2D her. Je vydán pod licencí zlib a podporuje platformy Windows, Linux, Mac OS X, iPhone, iPad a Android. Narozdíl od mého projektu je libGDX napsán v jazycích C a C++, stejně tak i aplikace pro něj se píší v těchto jazycích. [22]

Domovská stránka projektu: `http://orx-project.org/`

1.2.3 Corona SDK

Je programátorské rozhraní pro iPhone, iPad a Android založené na jazyce Lua. Podporuje práci s multimédii i formátem JSON. Toto SDK je ovšem komerčním produktem, licence pro jednu platformu se pohybuje kolem 200 dolarů ročně. [23]

Domovská stránka projektu: `http://www.anscamobile.com/`

1.2.4 Unigine

Jde o komerční řešení pracující na OpenGL i DirectX. Pracuje na stejných platformách jako ORX, tedy Windows, Linux, Mac OS X, iPhone, iPad a Android. Toto řešení je vhodné pro velké komerční projekty, čemuž odpovídá i cena za licenci (na webu uvádí přibližně 30 000 dolarů za jeden projekt). [24]

Domovská stránka projektu: `http://unigine.com/`

1.2.5 Unity

Unity je především 3D engine pro weby, Windows, Mac OS X a nově i Android a iOS. Umožňuje dokoupení podpory pro Nintendo Wii, PlayStation 3 a Xbox 360. Pro psaní aplikací používá UnityScript nebo C++, nesouvisí tedy s Javou. [25]

Domovská stránka projektu: `http://unity3d.com/unity/engine/`

1.3 Řešení nesourodosti platforem

1.3.1 Třídy pro práci s geometrickými útvary

V Javě existuje sada tříd a rozhraní pro práci se základními geometrickými útvary, jako jsou obdélníkové oblasti, body apod. Desktopová edice Javy má k tomuto účelu třídy z balíčku java.awt a java.awt.geom. Oproti tomu Android balíček *AWT* (až na malou výjimku, kterou je java.awt.font) postrádá. Místo toho používá vlastní třídy z balíčku android.graphics. Protože moje platforma musí fungovat na obou systémech bez rozdílu, bylo nutné vytvořit vlastní sadu tříd. Seznam obdobných nebo podobných tříd na jednotlivých platformách popisuje tabulka 1. [3, 8]

Java SE	**Android**	**Vlastní**
java.awt.Point	android.graphics.PointF	Vector
java.awt.Shape	android.graphics.Region	Area, EditableArea
java.awt.Rectangle	android.graphics.RectF	Rectangle
(nemá)	*(nemá)*	Points, EditablePoints
java.awt.geom.Path2D	android.graphics.Path	Polyline

Tabulka 1: Třídy na jednotlivých platformách

V případě třídy Polyline nejde o obdobu jako takovou, protože třídy na jiných platformách ukládají cestu jako posloupnost čar a křivek. Polyline ukládá pouze pozice bodů v rovině. Jelikož se má práce týká dvourozměrného vykreslování, nejsou zde žádné třídy pro práci s prostorovými útvary.

1.3.2 Další obecné třídy

Vedle již jmenovaných tříd pro práci s geometrií bylo nutné zapouzdřit i další prvky, potřebné pro práci s grafikou apod. Pro práci s barvou je zde třída Color, pro směr zase výčtový typ Direction.

1.3.3 Práce s formátem JSON

Protože moje platforma využívá formát *JSON* pro ukládání dat, bylo nutné vybrat nástroj pro jeho zpracování. Pro Javu existuje několik na sobě nezávislých implementací od různých autorů s různými možnostmi.

Protože je JSON používán i k ukládání zdrojů (např. tabulky řetězců, textury), může takový soubor obsahovat poměrně velké množství dat. Aby nemusel být načítán celý soubor se zdroji najednou (jak je tomu např. v XML při použití DOM), bylo by nutné použít JSON parser, který by umožňoval postupné načítání a zpracovávání JSON přímo z datového proudu. Protože technologie XML nabízí tzv. push a pull parsery (užívané převážně pouze pro čtení), hledal jsem obdobné projekty i pro JSON. Projekt *org.json* nenabízí streamovaná API, existuje ale nezávislé rozšíření *JSON.simple*, které přidává funkcionalitu push parseru (tedy že reagujeme na události generované parserem). *JSON.simple* je použit

v mnoha projektech, je výkonný a odzkoušený. Pro pull parsování je zase možné využít projekt *Jackson*, což je parser umožňující mimo jiné použít u JSON obdobnou techniku jako je StAX u XML. [1]

Stránky projektů:
`http://code.google.com/p/json-simple/`
`http://jackson.codehaus.org/`

Java SE pak žádnou možnost zpracování JSON v základu nenabízí. Oproti tomu Android obsahuje v základu balíček org.json, což je referenční implementace formátu JSON, kterou napsal zakladatel projektu Douglas Crockford. Projekt org.json se vyvíjí s otevřeným zdrojovým kódem na webu *GitHub.com*. Aktuální třídy (projekt není rozdělen na verze) se ale od těch Androidovských liší svým rozhraním - nejsou tedy kompatibilní. Pokud bych chtěl využít balíček org.json, musel bych přizpůsobit třídy tak, aby svým rozhraním odpovídaly těm v Androidu. Z tohoto důvodu, a případnému možnému rozšiřování mé platformy (nebo systému Android), jsem se rozhodl, že prozatím použiji vlastní třídy pro práci s formátem JSON (založené na zdrojových kódech balíčku org.json) bez dalších rozšíření. V současné verzi jde o zdrojové kódy přímo z *GitHub.com* ke dni 2. února 2012, přesunuté do balíčku buc0014.bp.json. Zachoval jsem následující třídy a rozhraní:

- buc0014.bp.JSONArray
- buc0014.bp.JSONException
- buc0014.bp.JSONObject
- buc0014.bp.JSONString
- buc0014.bp.JSONStringer
- buc0014.bp.JSONTokener
- buc0014.bp.JSONWriter

Ostatní třídy jsem z balíčku odstranil, jelikož je k práci na projektu nepotřebuji a neposkytuje je ani systém Android. Šlo převážně o třídy umožňující koverzi mezi formátem JSON a jinými formáty, jako je JSONML (formát JSON převedený do XML), hlavičkou protokolu HTTP nebo formátem CDL. [3, 12]

1.3.4 Java 6

Pro psaní aplikací je nutné použít jazyk Java verze 6, ve kterém je napsána i samotná platforma. Nejvyšší dostupná verze Javy v době tvorby této práce byla 7, která v rámci jazyka přináší několik změn. Přibývá zde možnost užít **switch** i s objekty typu String, rozhraní AutoCloseable nebo vylepšení odchytávání výjimek. Protože je ale Android vystavěn na Javě 6, chybí mu přidané třídy a rozhraní. Žádná z těchto změn taktéž není pro použití

mé platformy nijak významná. Z pohledu kompilace zdrojového kódu Javy pod JDK7 by žádný podstatný problém nastat neměl, jelikož byte-kód Javy nedoznal změn (vyjma přidání instrukce InvokeDynamic, jež je využívána skriptovacími jazyky s dynamickým typováním proměnných). Eclipse ale použití Android SDK v kombinaci s JDK7 vylučuje. Abych předešel dalším možným problémům, a také z důvodů výše uvedených, jsem se rozhodl omezit pouze na Javu 6. [7]

1.3.5 OpenGL

OpenGL je programové rozhraní grafického hardwaru. Obsahuje asi 200 příkazů a dalších 50 příkazů v doprovodné knihovně utilit. Jde o moderní rozhraní nezávislé na použitém zařízení a fungující na mnoha platformách. [2]

Pro přístup k akcelerovanému vykreslování grafiky se nabízí již jmenované OpenGL nebo rozhraní Microsoft DirectX (jeho část Direct3D, případně nové rozhraní XNA), které je závislé na platformě Windows (případně XNA pak na .NET frameworku). Oproti tomu rozhraní OpenGL je podporováno jak na systému Android, tak na operačních systémech běžných pro desktopové počítače (Windows, Linux, Unix apod.).

Systém Android nabízí podporu OpenGL mimo jiné skrze balíček *android.opengl*. Jelikož je Android primárně určen pro mobilní zařízení a využívá OpenGL ES[16] (OpenGL pro embedded systémy), omezil jsem možnosti mé platformy pouze na příkazy tohoto rozhraní. Zvolil jsem verzi 1.x pro hardware s pevně danou funkčností (fixed function hardware), což je pro rozsah mé práce dostačující. Verzi 2.0 s podporu shaderů jsem tedy do práce nezahrnoval.

Dalším omezením OpenGL ES je, že v případě čísel s pohyblivou řádovou čárkou pracuje pouze s jednoduchou přesností (v Javě jsou tato čísla reprezentována datovým typem **float**, rozsah 32 bitů podle IEEE 754). Z toho důvodu jsou všechny třídy v platformě přizpůsobeny pro práci s typem float. Datový typ **double** (dvojitá přesnost rozsahu 64 bitů) tedy není v platformě vůbec zastoupen ani podporován. Ze stejného důvodu je používán i typ **short** a buffery stejného typu. OpenGL ES má totiž vlastní seznam podporovaných datových typů[17], pro které jsem musel použít odpovídající typy v Javě, jak ukazuje tabulka 2. Jedna z funkcí rozhraní, kterou moje platforma používá, je glDrawElements[5]. Ta přijímá pouze jednobajtové nebo dvoubajtové číslo bez znaménka (konstanty GL_UNSIGNED_BYTE a GL_UNSIGNED_SHORT). Jelikož Java nepodporuje typ short bez znaménka, využívá platforma javovský typ **short** a to pouze v rozsahu 0 až 32767. Totéž platí pro typ **int**, který je v případě použití bez znaménka využit pouze v rozsahu 0 až (2^31)−1.

Na rozdíl od Androidu, Java SE v základu nenabízí přímý přístup k funkcím knihovny OpenGL. Řešením je použít některý z existujících projektů, který se touto problematikou zabývá. Knihovna *LWJGL* nabízí přístup nejen k OpenGL, ale také OpenCL a OpenAL. Nabízí tedy multimediální řešení schopné zpracovat i zvuk. K běhu ale vyžaduje plnou funkcionalitu knihovny OpenGL, nedovede pracovat pouze s OpenGL ES, tedy se nehodí pro Android. [14] Projekt *Java3D* (popsaný v JSR 926) nabízí balíčky schopné pracovat s vektorovou matematikou i vykreslováním obrazu. Jeho funkce jsou ale objektově orientované, pro přístup k OpenGL se mi zdál příliš komplexní. [6] Pro potřeby mé

platformy na Java SE jsem vybral *JOGL*, který poskytuje jednoduchý binding knihovny do Javy. Součástí JOGL je i rozhraní *NEWT*, které umožňuje vytvořit okno pro vykreslování přímo v použitém operačním systému (podobně jako GLUT). Tuto možnost jsem využil, abych se vyhnul nutnosti vytvářet okno skrze komponenty AWT nebo Swing. [10]

GL Typ	Bitů	Typ v Javě	Popis
boolean	1	boolean	boolean (logická hodnota)
byte	8	byte	celé jednobajtové číslo se znaménkem
ubyte	8	byte	celé jednobajtové číslo bez znaménka
char	8	byte	znak pro tvorbu řetězců
short	16	short	celé dvoubajtové číslo se znaménkem
ushort	16	short	celé dvoubajtové číslo bez znaménka
int	32	int	celé čtyřbajtové číslo se znaménkem
uint	32	int	celé čtyřbajtové číslo bez znaménka
fixed	32	*(nemá)*	čtyřbajtové číslo s pevnou řádovou čárkou
sizei	32	int	kladné čtyřbajtové číslo
enum	32	int	hodnota výčtu
intptr	ptrbits	*(nemá)*	celé číslo se znaménkem
sizeiptr	ptrbits	*(nemá)*	celé číslo se znaménkem
bitfield	32	int	bitové pole
float	32	float	čtyřbajtové číslo s pohyblivou řádovou čárkou
clampf	32	float	čtyřbajtové číslo s pohyblivou řádovou čárkou omezené na hodnotu 0 nebo 1

Tabulka 2: Datové typy OpenGL ES

1.4 Rozdíly v implementacích

Zatímco některé třídy jsou naimplementovány obecně a jsou přímo použitelné na desktopové Javě i v systému Android, zbylá část tříd využívá či zapouzdřuje platformově závislé třídy. Mezi ně patří třídy Environment, GLU, data.PNG, drawing.Font, drawing.Texture a třídy pro práci se zvukem.

1.4.1 Verze pro Java SE

První verze implementace, určená pro desktopovou Javu. Třída Environment využívá pro výstup proudy třídy System a pro vytvoření okna s OpenGL a zachytávání vstupu používá třídu StandartWindow, která je chráněná a nachází se pouze v této verzi implementace. StandartWindow zapouzdřuje třídy z balíčku javax.media.nativewindow a další prvky JOGL. Celé OpenGL je tedy realizováno pomocí JOGL. Nahrávání a ukládání formátu PNG ve stejnojmenné třídě je řešeno pomocí java.awt.image.BufferedImage a javax.imageio.ImageIO. Třída Font využívá Font z balíčku java.awt, včetně dalších pomocných tříd pro práci s písmem. Zvuk je řešen pouze provizorně pomocí balíčku javax.sound.sampled a bude zdokonalen v dalších verzích mé platformy.

1.4.2 Verze pro Android

Environment na systému Android používá pro výstup třídu android. util .Log, což usnadňuje ladění. Propojení s OpenGL je v tomto případě řešeno opět pomocí chráněné třídy, tentokrát je to ale AndroidActivity (potomek android.app. Activity) a AndroidView (potomek android.opengl.GLSurfaceView), který řeší i uživatelské vstupy. Obě třídy se nachází pouze v této verzi implementace. Přístup k funkcím OpenGL je realizován skrze balíček khronos, který je součástí mikroedice Javy, částečně v kombinaci s android.opengl. Formát PNG je na Androidu spravován skrze třídy android.graphics.Bitmap a android.graphics.BitmapFactory.

1.5 Vývoj aplikací

1.5.1 Pravidla vývoje

Pro použití mojí platormy platí následující pravidla:

- pokud existuje třída, která je součástí platformy a splňuje/poskytuje to, co potřebujete, použijte ji; pokud se tam nenachází, použijte třídu ze seznamu kompatibilních balíčků
- při práci s čísly s pohyblivou řádovou čárkou upřednostňujte float
- pokud potřebujete načíst data typu, který umí platforma obstarat pomocí jednotného načítání zdrojů, použijte tuto metodu
- je-li to možné, vyhněte se všem přímým závislostem na konkrétním hardwaru (tedy programujte tak, aby ošetření vstupu nebylo řešeno pouze použitím klávesnice nebo naopak pouze pomocí dotykového displaye); platforma je k tomu náležitě vybavena
- s ohledem na mobilní zařízení nevyužívejte přehnané množství systémových zdrojů (například se vyhněte texturám s rozlišením vyšším než 1024x1024 pixelů apod.)

1.5.2 Společné balíčky

Vedle balíčků a tříd specifických pro Android nebo desktopovou Javu existují i takové, které jsou společné pro obě platformy. Seznam dalších balíčků, použitelných při vývoji vlastní aplikace je popsán v tabulce 3. I přesto je možné, že některé třídy a rozhraní v balíčcích nemusí být shodné, např. z důvodu změn v Javě 6. Aby byl zajištěn bezproblémový běh výsledné aplikace, je možné použít několik řešení.

První možností by bylo sestavit nástroj kontrolující rozhraní zkompilovaných tříd pomocí reflexe jazyka Java a porovnávající ho s databází tříd, které jsou pro obě platformy společné. Součástí tohoto řešení by dále musel být nástroj, schopný pomocí reflexe projít rozhraní na obou platformách a vytvořit databázi obsahující průnik výsledků.

Druhou možností je začlenit do vývojářské verze mé platformy i vygenerovaná prázdná rozhraní tříd, obsahující pouze společné vlastnosti a metody pro obě platformy.

Úplná podpora		
java.io	java.util[.*]	javax.xml.transform.dom
java.lang	javax.crypto[.*]	javax.xml.transform.sax
java.lang.ref	javax.net	javax.xml.transform.stream
java.lang.reflect	javax.net.ssl	javax.xml.validation
java.math	javax.sql	javax.xml.xpath
java.net	javax.xml	org.w3c.dom
java.nio[.*]	javax.xml.datatype	org.w3c.dom.ls
java.security[.*]	javax.xml.namespace	org.xml.sax
java.sql	javax.xml.parsers	org.xml.sax.ext
java.text	javax.xml.transform	org.xml.sax.helpers
(*vyjma:* java.util.spi)		
Částečná podpora		
java.awt.font	javax.security.auth	javax.security.auth.x500
java.beans	javax.security.auth.callback	javax.security.cert
java.lang.annotation	javax.security.auth.login	

Tabulka 3: Společné balíčky

Po překladu pro konkrétní platformu tedy bude zajištěna plná funkčnost. I toto by vyžadovalo připravit příslušné nástroje pro generování těchto rozhraní.

Řešení tohoto problému ale není součástí této bakalářské práce. V případě dalšího vývojé mé platformy bude samozřejmě některé z řešení zahrnuto do projektu. [9]

1.6 Volba nástroje

Při volbě vývojového nástroje jsem se rozhodl pro *IDE*, která na rozdíl od jednoduchých textových editorů umožňují lepší integraci a práci s nástroji pro automatické sestavování aplikací. Důvodem bylo i to, že kompilace projektu napsaného pro mou platformu bude sama vyžadovat automatické sestavování v nástroji *Apache Ant*^{TM}. Ze svobodných IDE, určených k psaní projektů v Javě, jsem zvolil nástroj *Eclipse*. Ten oproti *NetBeans* nabízí plugin ADT sloužící k tvorbě projektů pro systém *Android*.[4]

1.7 Volba licence

Dalším krokem bylo zvolit pro práci vhodnou licenci. Platforma není naplánována jako komerční projekt, proto jsem vybíral pouze z licencí pro open-source a svobodný software. Licence jako *BSD*[26] a *MIT*[29] mi nevyhovovaly, protože dovolují začlenění zdrojového kódu do komerčního projektu bez nutnosti jakékoli zmínky o tom. Často používaná licence *GPL*[27] zase vyžaduje licencovat odvozené dílo jako *GPL*, což není vhodné pro knihovny, a tedy ani mou platformu. Zvolil jsem proto licenci *LGPL*[28], která umožňuje sloučení do projektu, který není licencován jako *GPL*[18].

2 Návrh a API

Při návrhu své platformy jsem kladl důraz na jednoduchost a snadnou pochopitelnost všech rozhraní, tříd a dalších mechanismů. Tato kapitola popisuje jejich návrh a použití.

2.1 Zdroje

Důležitou částí platformy je systém, starající se o jednotné nahrávání zdrojů. Android využívá k tomuto účelu třídu R, jedná se ale o platformově závislé řešení. Taktéž přímý přístup k souborům, který umožňuje desktopová Java i Android, není úplně ideální, protože tvary URL se na obou platformách liší. Navíc v případě Androidu je nutné získat si cesty k systémovým složkám přes specifické rozhraní, konkrétně přes metody jeho třídy Environment. Proto jsem se rozhodl pro vlastní řešení pomocí pojmenovaných zdrojů. Přístup k nim je řešen přes hlavní třídu platformy Environment a mapování jmen na soubory řeší konfigurační soubory resp. soubory se zdroji. Konfigurační soubory budou textové, čitelné a upravitelné jak strojově, tak uživatelem. Proto jsem musel zvolit vhodný formát. [1]

2.1.1 XML

XML je rozšiřitelný značkovací jazyk odvozený od jazyka SGML. Jde o standart W3C a v dnešní době patří k nejdůležitějším formátům výměny dat strukturovaným způsobem. Výhody jazyka XML jsou následující:

- data jsou zapsána bez ohledu na platformu, jsou plně přenositelná
- Java má v základu nástroje pro zpracování XML
- lze použít libovolné značky popsané v šabloně
- je vyřešeno kódování textu díky hlavičce XML dokumentu

Mezi nevýhody lze započítat o několik řádů větší velikost souboru oproti binárnímu způsobu uložení dat. Přestože tímto problémem trpí i další textové formáty, v případě XML je rozdíl oproti binárnímu ukládání významnější především při ukládání polí, kdy každý prvek musí být reprezentován jedním elementem. Řešením je uložit pole do formátu CSV a ten pak vložit jako text do XML elementu, což ale není vhodné řešení z pohledu XML. Výsledný dokument pak nebude čitelný bez nutnosti řešit parsování této části a jiné programy nebudou schopny ani pomocí šablony XML pochopit význam dokumentu - přišel by tedy o nezávislost a přenositelnost.

Základem formátu XML je dokument, což může být soubor, proud dat apod. Obsah dokumentu je popsán pomocí značek, přičemž v každém dokumentu XML musí existovat právě jeden tzv. kořenový element. Ostatní elementy se nazývají vnořené. Element samotný je pak reprezentován počáteční a koncovou značkou včetně obsahu uloženého mezi značkami (element může být i prázdný). Elementy dále obsahují atributy, což je dvojice *název="hodnota"* umístěná do počáteční značky. Znaky použitelné v názvech elementů i atributů jsou v tomto případě omezeny standardem XML. [1]

2.1.2 JSON

Jde o odlehčený formát pro výměnu dat, který je snadno čitelný i zapisovatelný člověkem. Současně jej lze jednoduše zpracovávat či vytvářet strojově. Celý formát je vlastně podmnožinou programovacího jazyka JavaScript (podle normy ECMA-262 3rd Edition - December 1999). Stejně jako XML je textový a na jazyce zcela nezávislý. JSON dále využívá konvence vzešlé z jazyka C, jelikož je založen na dvou základních strukturách:

- kolekce uchovává data ve formě *název: hodnota* a různé jazyky mají pro tuto strukturu různé způsoby realizace, např. objekt, záznam (record), struktura (struct), slovník, hash tabulka nebo asociativní pole
- tříděný seznam hodnot uchovává data s ohledem na pořadí, přičemž každá hodnota má přidělen index, v programovacích jazycích se pak reprezentuje pomocí pole (array), vektrou, seznamu (list) nebo sekvence

Jedná se o obecné struktury, které jsou ve většině moderních programovacích jazyků podporovány. JSON pro ně tedy zavádí obecný výměnný formát zápisu.

Nevýhodou formátu oproti XML je, že neřeší uložení do souboru ani kódování textu. Řetězce v JSON jsou sice posloupnosti znaků sady UNICODE, ale formát nemá vlastní hlavičku a není tedy možné jej jednoznačně rozpoznat bez nutnosti parsování obsahu.

Formát je popsán v RFC 4627 jako MIME typ *application/json*.[11]

2.1.3 YAML

YAML (YAML Ain't Markup Language) je textový formát pro serializaci dat. Formát klade důraz na jednoduchou čitelnost člověkem, oproti JSON však vyžaduje složitější parsování. Dvě základní struktury YAML jsou *sekvence* a *mapy* (obdoba pro pole a objekty v programovacích jazycích). Pro tyto struktury existuje více způsobů zápisu:

- *sekvence* je možné zapisovat jako odrážkový seznam, kdy jedna hodnota odpovídá jednomu řádku, který začíná na znak spojovníku (-) a blok může být odsazen mezerami, což mění význam dat; druhou možností je použít zápis, kdy se sekvence uzavře do lomených závorek a hodnoty se oddělí mezerami, jako je tomu v případě JSON
- *mapy* se zapisují také na řádky, přičemž *jméno* je od *hodnoty* odděleno znakem dvojtečky; opět lze použít i zápis podobný JSON pomocí složených závorek

Narozdíl od JSON nebo XML využívá YAML bílé mezery k odsazování a definování bloků dat. Taktéž využívá konců řádků, což lze v rámci dokumentu ovlivnit různými značkami. Nedovede však zpracovat tabulátory, ty musí být nahrazeny mezerami, jinak skončí parsování chybou.

Formát je ve verzi 1.2 a nabízí mnoho zajímavých prvků, jako např. rozlišení datových typů hodnot, možnost jejich přetypování nebo aliasy pro jména hodnot. Umožňuje taktéž přímo uložit bloky binárních dat, k čemuž má speciální značku.

Data je dále možné dělit na dokumenty, ovšem chybí zde typická hlavička pro rozpoznání formátu v případě souboru. Pro kódování v YAML se používá sada UNICODE a UTF-8 nebo UTF-16. Na konfigurační soubory v mém projektu mi však přišel YAML zbytečně složitý. [19, 20]

2.1.4 Vlastní formát

Poslední možností bylo navrhnout vlastní textový formát. Nabízelo se odvození od formátu INI používaného v operačním systému Windows, což je soubor čtený po řádcích rozdělený na sekce. Řádky pak obsahují data ve formě klíč-hodnota oddělená znakem *rovná se*. Jakékoliv podobné řešení má ale několik podstatných nedostatků:

- je nutné vymyslet jednoduchý formát nabízející přiměřené možnosti použití
- programátoři by byli nuceni se učit nový syntax a práci s tímto formátem
- bylo by nutné řešit vlastní specifikaci takového formátu a psát jeho parser

Z tohoto důvodu jsem řešení vlastním formátem zamítl.

2.1.5 Shrnutí

Z vybraných formátů jsem nakonec zvolil formát JSON, který nejvíce vyhovoval mým požadavkům. Problém s absencí hlavičky a nemožnosti nastavit kódování jsem vyřešil tím, že výchozí kódování textových souborů pro mou platformu je UTF-8. Platforma ani jiné kódování nepodporuje (i když Java ano), což s ohledem na rozšíření a použitelnost sady UNICODE není podstatné.

2.2 Aplikace

2.2.1 Zavaděč aplikace

Aplikace pro moji platformu je chápána jako souhrn tříd, zdrojů a konfiguračních souborů, včetně zaváděcího souboru. Ten se zpravidla jmenuje *start.json* a může být umístěn ve složce souborového systému nebo zabalen v souboru formátu ZIP spolu s celou aplikací. Zaváděcí soubor je, stejně jako všechny konfigurační soubory a soubory se zdroji, formátu JSON. Kořenovým prvkem je zde objekt, který popisuje vlastnosti aplikace. Dostupná sada vlastností, včetně výchozích hodnot, je uvedena v tabulce 4.

Jakmile je zaváděcí soubor načten a zpracován, načtou se požadované soubory se zdroji a vytvoří se instance požadované třídy, která bude tvořit základ celé aplikace. Aplikace poběží v okně (jen v případě desktopové Javy) nebo celoobrazovkově. Rozměry a chování okna pro desktopovou Javu je možné upřesnit v objektu *window*, jehož vlastnosti popisuje tabulka 5.

Název	Typ	Výchozí	Popis
name	řetězec	Nameless	Jméno aplikace (např. pro popisek okna)
tick	řetězec	1	Rychlost časovače (0 pro vypnuto)
class	řetězec	*(povinné)*	Hlavní třída aplikace (s rozhraním Application)
resources	pole	[]	Seznam souborů se zdroji
window	objekt	{}	Vlastnosti okna (viz. níže)

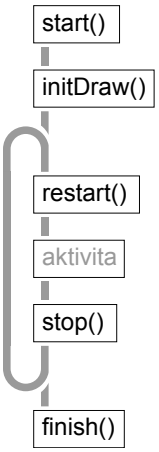
Tabulka 4: Vlastnosti aplikace

Název	Typ	Výchozí	Popis
width	číslo	640	Šířka plochy v pixelech
height	číslo	480	Výška plochy v pixelech
fullscreen	boolean	false	Celoobrazovkový režim
minimum-ratio	číslo	0	Minimální poměr stran (0 vypnuto)
maximum-ratio	číslo	0	Maximální poměr stran (0 vypnuto)

Tabulka 5: Vlastnosti okna aplikace

2.2.2 Rozhraní a životní cyklus

Jakmile je aplikace spuštěna, platforma s ní komunikuje pomocí metod rozhraní Application. Životní cyklus aplikace je popsán diagramem na obrázku 1.



Obrázek 1: Životní cyklus aplikace

Metoda start se volá jako první a pouze jednou. Následuje volání metody initDraw, kdy je předán kontext OpenGL a aplikace si může libovolně upravit prvotní hodnoty. Při každé změně nastavení, při startu aplikace a také při znovuobnovení běhu aplikace je volána metoda restart. Ta nese v parametru nové hodnoty nastavení, na které je možné aplikaci adekvátně připravit (např. upravit rozlišení). Po zavolání metody restart se aplikace stává *aktivní*, tedy jsou volány metody draw a tick. Metoda draw je volána pokaždé,

když je nutné aplikaci překreslit. Metoda tick je volána asynchronně vůči draw, jde o hlavní časovač aplikace. Pokud dojde k přerušení běhu, např. z důvodu minimalizace okna na desktopovém systému nebo uzavření aplikace na systému Android, zavolá se metoda stop. Tato metoda se volá pouze, pokud je aplikace v *aktivním stavu* a po dokončení volání přechází do stavu neaktivního. K obnovení aktivity je nutné opětovné volání metody restart. Pokud má dojít k úplnému vypnutí aplikace (což je možné pouze v neaktivním stavu), dojde k zavolání metody finish a následně k úplnému ukončení běhu platformy.

2.2.3 Definice zdrojů

Formát souborů se zdroji je podobný zaváděcímu souboru. Kořenovým elementem je zde také objekt, tentokrát má ale pouze vlastnost *resources*, která je rovněž typu objekt. Jednotlivé vlastnosti objektu *resources* jsou pak přímo jména zdrojů a jejich hodnoty data zdrojů. Data jsou opět typu objekt a jedinou povinnou vlastností je řetězec *type*, určující druh zdroje. Typy zdrojů, které dovede platforma načítat a ukládat, popisuje tabulka 6. Další vlastnosti objektu jsou většinou metadata. Samotná data, jejichž typ a způsob uložení závisí právě na metadatech, jsou uložena ve vlastnosti *data*.

Název	**Třída**	**Popis**
area	buc0014.bp.core.Area	Obdélníková oblast
color	buc0014.bp.core.Color	Barva bez alfa kanálu
font	buc0014.bp.drawing.Font	Písmo včetně vlastností
points	buc0014.bp.core.Points	Uspořádaná množina bodů
sequence	buc0014.bp.core.Sequence	Sada uspořádaných množin bodů
strings	java.util.Map<String, String>	Sada pojmenovaných řetězců
texture	buc0014.bp.drawing.texture	Obrázková textura
texture-map	buc0014.bp.core.TextureMap	Sada výřezů pro mapování textury
tiles	buc0014.bp.drawing.Tiles	Dvourozměrná sada dlaždic

Tabulka 6: Typy zdrojů

Typy *area*, *points* a *sequence* mají vlastnosti *width* a *height* (výchozí hodnota pro obě je 1). Během načítání dat je pak každá hodnota vydělena příslušnou vlastností šířky nebo výšky. Tato vlastnost zjednodušuje zadání např. pro výřezy textur, kdy je výřez definován v rozmezí 0.0 až 1.0 pomocí desetinných čísel. Pro zjednodušení tedy stačí hodnoty *width* a *height* nastavit na skutečný rozměr textury a data lze pak zadávat přímo jako souřadnice v pixelech, což je lépe čitelné i představitelné, než čísla s mnoha desetinnými místy.

Oblast (typ *area*) má *data* definována jako pole se čtyřmi hodnotami, jmenovitě *x0*, *y0*, *x1* a *y1*, což jsou absolutní souřadnice jednotlivých hran obdélníkové oblasti.

V případě *points* jsou *data* rovněž typu pole, přičemž musí mít sudý počet prvků. Hodnoty jsou poté načítány postupně jako x a y pro jednotlivé, po sobě jdoucí body. U *sequence* jde pak o dvourozměrné pole, kde první pole obsahuje jednotlivé množiny bodů a vnořená pole pak reprezentují samotné *points*.

Typy *font*, *strings* a *texture-map* nemají jako jediné typy žádné vlastnosti navíc, obsahují pouze *data*. V případě *strings* jsou pak *data* objekt, který přímo odpovídá mapě řetězců.

Pro zapsání barvy (typ *color*) lze použít několik formátů, které lze nastavit pomocí vlastností z tabulky 7 (výchozí hodnoty vlastností jsou vypsány tučně). Čísla, která reprezentují barevné složky, lze zadávat jako desetinná nebo v rozsahu jednoho bajtu. Na výběr je ze tří barevných modelů.

Název	Hodnoty	Popis
precision	**float**	Rozsah 0.0 až 1.0
	byte	Rozsah 0 až 255
model	**rgb**	Model Red-Green-Blue
	hsl	Model Hue-Saturation-Light
	hsv	Model Hue-Saturation-Value

Tabulka 7: Vlastnosti barvy

Textury mohou být načítány z vlastních souborů formátu PNG, nebo být uloženy přímo v souboru se zdroji. Toto nastavení je možné měnit pomocí vlastností textur, popsaných v tabulce tabulce 8 (výchozí hodnoty vlastností jsou vypsány tučně). Ve všech případech jsou *data* uložena jako řetězec.

Název	Hodnoty	Popis
encoded	**false**	Data obsahují jméno PNG souboru
	true	Data obsahují přímo data textury
encoding	**base64**	Podporováno pouze kódování Base64
compression	**no**	Data nejsou nijak komprimována
	gzip	Data jsou komprimována metodou GZIP
	deflate	Data jsou komprimována metodou DEFLATE

Tabulka 8: Vlastnosti textury

Posledním typem je typ *tiles*, což je dvourozměrná mapa dlaždic. Má vlastnosti *width* a *height*, které v tomto případě udávají rozměry mapy. Pokud jsou *data* typu *string*, lze nastavit i vlastnosti *encoding* a *compression*, které již byly zmíněny v tabulce 8. Data mohou být dále uložena i ve formě dvourozměrného pole, což je nepraktické z pohledu uchování, zato je možné provádět na datech ruční úpravy.

2.3 Balíček platformy

Balíček s označením buc0014.bp obsahuje základní třídy a prvky pro běh celého projektu, včetně zavaděče. S ohledem na implementaci může obsahovat i několik chráněných tříd použitých k vnitřnímu chodu platformy.

Nejdůležitější (zaváděcí) třídou je třída Environment. Jde o obdobu třídy System v desktopové edici Javy nebo Environment v systému Android. Třídu nelze instancializovat, všechny operace jsou řešeny pomocí statických metod. Její funkčnost lze rozdělit do několika základních bloků:

- **informační** metody slouží k zjištění informací o platformě a aplikaci, konkrétně lze získat jméno, verzi, web a další informace popsané dále v práci včetně údajů o displayi
- **výstupní** metody pro logování důležitých informací aplikace, které nahrazují práci se System.out a System.err
- metody pro práci s **uživatelským vstupem** umožňují zjišťovat pozici ukazatele nebo použít přímý vstup pro vkládání textu
- načítání informací i dat **zdrojů** z příslušných souborů
- přístup k **úložišti** je řešen taktéž přes tuto třídu

V balíčku platformy se nachází i rozhraní GL a třída GLU pro práci s OpenGL. Dále je zde pomocná třída ScreenDescriptor jako kontejner pro vlastnosti displaye, ResourceLoader pomáhající v asynchonním nahrávání zdrojů a anotační typ Version pro definici verze platformy i aplikací. Posledním a nejdůležitějším prvkem je rozhraní Application, které tvoří kostru každé aplikace pro moji platformu.

2.4 Základní balíček

Třídy a rozhraní pro práci se základní geometrií a dalšími obecnými prvky jsou v balíčku buc0014.bp.core. Implementace balíčku je nezávislá na systému a je pro desktopovou verzi Javy i Android stejná. Obsahuje např. třídy pro zpracování obdélníkové oblasti, bodů a barev, jak již bylo popsáno výše. Bližší informace o implementaci jsou uvedeny v samostatné kapitole, která se tomuto balíčku tříd věnuje.

2.5 Ovládací balíček

O ošetření uživatelského vstupu včetně textu se stará balíček buc0014.bp.controlling. Jsou zde pomocná rozhraní pro informace o bodovém ukazateli, přímý textový vstup i výchozí implementace pro tyto interfacy. Tato rozhraní jsou využívána v kombinaci s třídou Environment z balíčku platformy.

2.6 Datový balíček

Balíček buc0014.bp.data obsahuje třídy pro načítání a zpracování dat platformy. Jeho součástí jsou pomocné interfacy, které využívá třída Environment k poskytování přístupu ke zdrojům, třídy pro ukládání a načítání komprimovaných dat (metody DEFLATE, GZIP a *Base64*) skrze datové proudy a také obsahuje podporu pro formáty PNG, TMX a JSON. Třída JSON umožňuje i serializaci a deserializaci všech obecných prvků ze základního balíčku, včetně obrazových dat textur apod.

2.7 Grafický balíček

Objektovou nástavbu nad OpenGL a celou grafickou část tvoří balíček buc0014.bp.drawing. Třídy tohoto balíčku se starají o vykreslování polygonů, správu textur a vykreslování písma. Polygon je třída, umožňující vykreslit libovolný n-úhelník včetně nastavení výřezu textury a barev jednotlivých vrcholů. Textura je reprezentována samostatnou třídou Texture a podporuje jak načítání z formátu PNG, tak vkládání výřezů. Potomkem je pak třída Font, která je schopna na texturu vykreslovat znaky zvoleného písma. Jsou zde i pomocné třídy pro nastavování stylů písma, které podporují i dědičnost vlastností textu.

2.8 Zvukový balíček

Poslední z balíčků projektu, buc0014.bp.playing, slouží pro zpracování zvuku. Zvuk byl původně plánován jako samostatý projekt, ale nakonec se stal součástí této práce. Protože je ale hlavní důraz kladen na grafickou část, je zde zvuk pouze provizorně. Bližší popis je na konci kapitoly 6.

3 Implementace balíčku platformy

Třídy tohoto balíčku jsou základem celého projektu. Nejdůležitější třída je Environment, která v případě desktopové verze obsahuje vstupní bod, metodu main. V této implementaci je dále několik chráněných metod, které se starají o zohlednění vstupních parametrů a načtení celé platformy. Při spouštění mé platformy je možné uvést jako první parametr jméno zaváděcího souboru. Pokud není explicitně uvedeno, použije se jako výchozí *start.json* v aktuální složce. Mohou následovat i další, upřesňující parametry, které popisuje tabulka 9.

Parametr	**Popis**
-h –help	Vypíše nápovědu o parametrech
-s –setup	Zobrazí dialog s nastavením
-f –fullscreen	Vynutí celoobrazovkový režim
-w –window	Vynutí řežim v okně (vyšší priorita)
-a –always-on-top	Vždy nahoře
–no-always-on-top	Klasické zobrazení (vyšší priorita)
–width=(int)	Šířka okna
–height=(int)	Výška okna

Tabulka 9: Seznam parametrů platformy

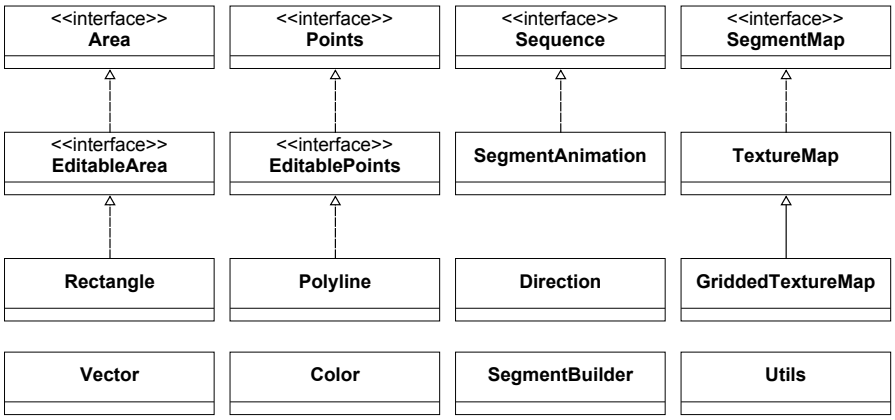
Jakmile je zaváděcí soubor načten a zpracován, vytvoří platforma odpovídající instanci pomocné třídy StandartWindow. Tato třída zaštituje funkčnost rozhraní NEWT. Jde o součást JOGL, umožňující spravovat nativní okno na daném operačním systému a renderovat do něj pomocí OpenGL. Toto řešení se pro mou platformu ideálně hodilo, protože je nezávislé na systému komponent. Nebylo tudíž nutné vytvářet okno pomocí sady AWT nebo Swing. [15]

V případě Androidu je vstupním bodem platformy třída AndroidActivity. Ta se postará o ošetření základních událostí, jako jsou pozastavení běhu platformy apod. Dále vytvoří instanci třídy AndroidView a zbytek práce nechá na ní. AndroidView vytvoří časovač a řídí vykreslování na plátno, které sama představuje. Je skrze ni řešen i uživatelský vstup. [1]

Ostatní třídy balíčku jsou velice jednoduché a jejich implementaci není třeba zvláště popisovat.

4 Implementace základního balíčku

Balíček jménem buc0014.bp.core obsahuje třídy a rozhraní všech základních primitiv potřebných pro práci s platformou. V této kapitole se věnuji bližšímu popisu těchto tříd a vysvětlení jejich vlastností. Seznam všech tříd balíčku včetně vztahů mezi nimi je vidět na obrázku 2.



Obrázek 2: UML schéma tříd základního balíčku

4.1 Obdélníkové oblasti

Důležitým prvkem pro vykreslování dvourozměrné grafiky jsou obdélníkové oblasti. Java SE i Android mají ke zpracování takové oblasti pouze jednu konkrétní třídu (která dědí přímo z Object a nemá žádné implementované rozhraní). Protože moje platforma klade na práci s grafikou velký důraz, rozhodl jsem se, že rozdělím práci s obdélníky a oblastmi na několik částí:

- Area - rozhraní pro získávání obdélníkové oblasti
- EditableArea - rozhraní pro práci s obdélníkovou oblastí
- Rectangle - třída pro práci s obdélníkovou oblastí

Pro práci s obdélníkem stačí použít třídu Rectangle. Výhodou rozdělení na rozhraní je, že jej mohou využívat i další třídy, které mají s obdélníkovou oblastí nějakou souvislost. Dobrým příkladem je třída Polyline, která slouží jako úložiště bodů. Protože jsou jednotlivé body seřazeny a očíslovány, lze se na objekt této třídy dívat jako na lomenou čáru. Jelikož tato čára zabírá určitou obdélníkovou oblast, implementuje třída Polyline rozhraní Area, díky čemuž lze zjistit velikost takové oblasti. Další využití je popsáno v kapitole o grafickém balíčku.

4.2 Práce s body a výřezy

Při práci s body jsem se rozhodl pro stejné členění, jako u obdélníkových oblastí.

- Points - rozhraní pro čtení jednotlivých bodů
- EditablePoints - rozhraní pro práci s jednotlivými body
- Polyline - třída pro práci s uspořádanou množinou bodů

Součástí projektu platformy původně nebyla třída pro práci s jedním bodem, protože prezentovat každý bod jako objekt mi přišlo pro práci s velkým množstvím bodů nevhodné. Později jsem do práce doplnil třídu Vector, která reprezentuje bod se souřadnicemi x a y. Tyto souřadnice jsou řešeny jako veřejné atributy, nejsou nijak chráněny proti změně nebo zápisu, a to z důvodu rychlosti.

Při definici výřezů textur je používáno právě rozhraní Points, jelikož výřez může být n-úhelníkový. Pro mapování těchto výřezů je zde pomocné rozhraní SegmentMap, které umožňuje spravovat pojmenované výřezy nebo sekvence. Výchozí implementací je třída TextureMap, která má i rozšíření pro textury rozdělené rovnoměrnou mřížkou, zvané GriddedTextureMap. Sekvence výřezů je poté reprezentována rozhraním Sequence a jeho výchozí implementací, třídou SegmentAnimation, která slouží k uchování sprite animací.

4.3 Barvy a ostatní

Pro uchování barvy používám vlastní třídu, který má jednotlivé barevné složky modelu RGB reprezentovány jako veřejné atributy. Složky jsou typu **float** v rozmezí 0 až 1 včetně, což usnadňuje zadávání do OpenGL. Hodnoty složek jsou dále neměnné, proto je nutné pro každou barvu konstruovat nový objekt. Současně třída poskytuje sadu základních barev ve formě symbolických konstant.

Vedle tříd je zde také výčtový typ Direction pro určení směru. Obsahuje 8 symbolických konstant pro základní směry v rozestupu 45 stupňů.

Poslední třídou balíčku je třída Utils, která slouží jako pomůcka pro práci se základními typy. Nemá veřejný konstruktor, nabízí pouze veřejné statické metody pro jednodušší porovnávání nebo tisk hodnot.

5 Implementace datového balíčku

Datový balíček se stará o načítání a ukládání dat. Obsahuje mimo jiné 3 třídy pro práci s kódováním Base64 a výčtový typ Compression, který je sám schopen vybranou kompresi aplikovat. K tomu slouží metody encode a decode, pracující s bajtovým polem.

5.1 Base64

Při ukládání většího množství dat ve formě textu (z důvodu použití formátu JSON) jsem se rozhodl používat kódování Base64. Třídy Base64InputStream a Base64OutputStream umožňují přímé zakódování nebo odkódování Base64 přímo v datovém proudu. Pro správnou funkci Base64OutputStream je s ohledem na implementaci nutné datový proud vždy po ukončení přenosu uzavřít. Třída Base64 pak nabízí metody pro převod pole bajtů a řetězců do nebo z tohoto kódování. Toto rozdělení jsem zvolil proto, že je vhodné pro různé situace, kdy je nutné Base64 použít.

5.2 Obrazová data

Další třída v balíčku je ImageData, což je jednoduchý kontejner na obrazová data. Používá se převážně při práci s texturami, jelikož uchovává nekomprimovaný obraz ve formě po sobě jdoucích pixelů, rozložených do pole bajtů. Vedle výšky a šířky obrázku umožňuje zjistit i formát uložení pixelů. ImageData poskytuje dva druhy uložení pixelů: s alfa kanálem a bez něj. Při použití alfa kanálu je každý pixel rozložen na čtyři bajty, kdy každý reprezentuje jednu složku modelu RGBA (v tomto pořadí). Bez alfa kanálu jsou bajty tři a každý uchovává jednu ze složek RGB modelu.

5.3 Přístup k datům

5.3.1 Rozhraní Loader

Vedle tříd se zde nachází i tři důležitá rozhraní: Loader, Source a zděděný Storage. První z nich, Loader, umožňuje rozšířit schopnosti mé platformy o další načítaný typ zdroje. Jelikož se předpokládá, že bude nutné uložit mezi zdroje JSON i formát, kterému moje platforma nebude rozumět, zavedl jsem toto rozhraní, obsahující dvě jednoduché metody:

```
public interface Loader
{
    public boolean canLoad(String type);

    public Object load(JSONObject object, Source source) throws JSONException;
}
```

Platforma si nejprve pomocí canLoad ověří, zda daný loader dovede daný typ načíst. Pokud ano, použije metodu load, které předá JSON objekt, získaný ze souboru zdrojů,

a objekt pro přístupu k úložišti. Toto rozhraní je používáno třídou ResourceLoader ze základního balíčku, která nahrává zdroje. Jeho základní implementací je pak třída JSON, která dovede načítat a ukládat všechny základní typy zdrojů.

Rozhraní Source je pak používáno třídou Environment k obecnému přístupu k úložišti, ovšem pouze pro čtení. Jde opět o velice jednoduché rozhraní, umožňující ověření existence pojmenovaného zdroje a vyžádání datového proudu pro čtení z něj:

```
public interface Source
{
    public boolean isSourceExists(String name);

    public InputStream openSource(String name) throws IOException;
}
```

Rozhraní Storage jej pak rozšiřuje o možnost zápisu:

```
public interface Storage extends Source
{
    public OutputStream openStorage(String name) throws IOException;
}
```

5.3.2 Další formáty

Vedle již zmíněných tříd jsou zde ještě další dvě třídy pro načítání formátů PNG a TMX. Třída PNG nabízí veřejné statické metody, které umožňují převod datového proudu na objekt třídy ImageData a naopak. Třída TMX zase reprezentuje mapu formátu TMX z editoru map Tiled, ze které lze načítat jednotlivé vrstvy dlaždic jako objekty třídy Tiles.

6 Implementace grafického a zvukového balíčku

Klíčovou částí mého projektu je balíček pro práci s grafikou, který zahrnuje i zpracování a vykreslování písma. Obecně existuje několik základních přístupů pro práci s písmem, jejichž popisu se budu nyní věnovat.

6.1 Zpracování textu

Písmo lze v OpenGL vykreslovat buď vektorově, nebo bitmapově. Při vektorovém vykreslování se každý znak složí z jednotlivých trojúhelníků a grafický adaptér pak provede rasterizaci. Druhý způsob spočívá ve vykreslení znaků na bitmapu, která se pak jako textura nanáší na nejčastěji obdélníkové polygony, kdy jeden takový polygon představuje právě jeden znak textu. Znaky můžou být na předpřipravené textuře nebo je možné je na texturu dynamicky dokreslovat podle potřeby. S ohledem na sadu Unicode jsem se rozhodl pro poslední zmíněné řešení.

Samotné vykreslování písma pak řeší třída Font. Při konstrukci objektu tohoto typu je nutné zadat jméno písma a velikost, ve kterém bude na texturu vykreslováno. Vzniklý objekt má pak všechny potřebné metody pro zjišťování rozměrů znaků, vykreslování znaků na texturu nebo provádění zarovnávání textu, včetně kerningu. Pro zadání vlastností vykreslovaného textu používám rozhraní FontStyle, jehož kód je následující:

```
public interface FontStyle
{
    public Color getColor();

    public float getSize();
}
```

Písmo je vykreslováno na texturu bíle. Pro získání jiné barvy se používá obarvování vrcholů polygonu, na který se výřez se znakem nanáší. Velikost v tomto případě udává taktéž velikost polygonu, znak zůstává na textuře vykreslen stále stejně velký. To může při špatném použití způsobit, že znak bude rozmazaný nebo budou zřetelné jednotlivé pixely. Je proto nutné, aby byla třída správně používána.

Třída TextStyle, která rozhraní FontStyle implementuje, definuje styl písma, včetně zarovnání. Objekt této třídy může dědit své vlastnosti po jiném objektu stejného typu, díky čemuž je možné vytvářet pojmenovanou hierarchii stylů.

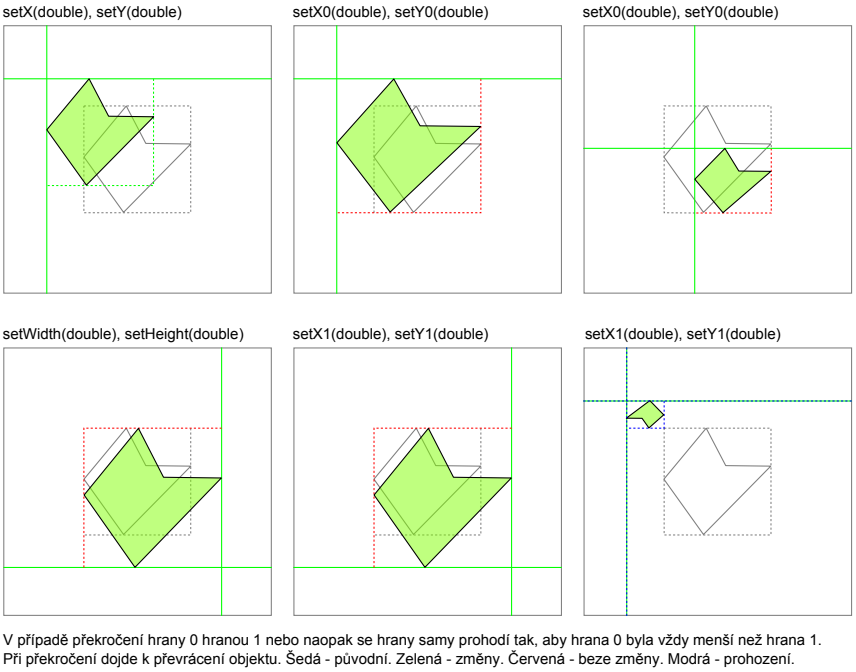
Samotné vykreslování textu je pak možné provádět třemi způsoby:

- použít komponentu TextLine, která vykresluje jeden nezalomitelný řádek textu s ohledem na bod
- použít komponentu TextBlock, která vykresluje zalomitelný odstavec textu s ohledem na obdélníkovou oblast
- vytvořit si vlastní řešení za použití metod třídy Font

6.2 Vykreslování polygonů

Polygony jsou vykreslovány stejnojmennou třídou Polygon. Ta umožňuje vykreslit libovolný n-úhelník, kdy je každému vrcholu možné nastavit vlastní barvu, pozici i uchycení textury. Vykreslení je pak provedeno pomocí funkce glDrawElements a bufferů.

Třída také disponuje rozhraním EditableArea, které má v tomto případě specifický vliv na objekt a body polygonu. Význam jednotlivých metod a jejich vliv na polygon je znázorněn na obrázku 3.



Obrázek 3: Vliv změn prováděných na polygonu přes rozhraní EditableArea

6.3 Zvukový balíček

Zvukový balíček je v práci zastoupen pouze částečně, jelikož jeho implementace nebyla pro práci klíčová. Rozvoj tohoto balíčku jsem naplánoval až pro případný další rozvoj projektu. Samozřejmostí by pak bylo přehrávání jednoduchých zvuků nebo streamování hudby. Projekt by také podporoval formát souboru OGG a kodek Vorbis.

Samotné rozhraní by se pak skládalo, podobně jako rozhraní pro obraz, z několika jednoduchých tříd. Třída Sound by se zachovala, přibyla by třída pro streamové zpracování hudby. Třída Player by byla rozšířena o nastavování kanálů a hlasitosti.

7 Ukázkové aplikace

Aby bylo možné funkčnost platformy náležitě otestovat, sestavil jsem 4 jednoduché aplikace. Platforma sice není plně dokončena, ale pro potřeby testování je stav práce dostačující.

První aplikace vykresluje postavičku na pozici kurzoru (v případě Androidu na pozici dotyku), čímž demonstruje ovládání vstupu. Postavička se stále drží ukazatele a při stisku tlačítka (ťuknutí na display) dojde ke změně postavičky. Rozdělení textury je řešeno pomocí komponenty GriddedTextureMap z balíčku core. Současně dojde k přehrání zvukového signálu. Dalším prvkem je animace pomocí spritů, postavička se stále lehce hýbe.

Obrázek 4: Screenshoty ukázkových aplikací

Druhá demonstrativní aplikace používá ve větší míře časovač. Jde o digitální kostku, kdy první dotyk zapne hod (kostka se točí, což signalizuje animace v rohu) a mění se hodnota čísla na kostce. Druhý dotyk hod kostky zastaví.

Třetí aplikace se věnuje vykreslování textu, konkrétně řeší zalamování do odstavců. Jde o test komponenty TextBlock.

Poslední aplikace testuje textový vstup. Jde o jakousi formu textové komponenty či promptu, kam je možné psát. Psaní je dostupné včetně mazání, pohybu kurzoru v textu a systémové schránky (pomocí klávesových zkratek), pokud je dostupná (desktopová Java).

8 Závěr

Výsledkem mé práce je použitelná multimediální platforma, která zaštiťuje funkčnost pro desktopovou Javu i systém Android. Implementace je zatím pouze ve fázi beta, ale již nyní je možné tvořit na ní plnohodnotné aplikace. Výhodou je, že pro přenos aplikace mezi různými zařízeními není nutný žádný zásah do jejích zdrojových kódů. Přenos na danou platformu je proveden již v době kompilace, i když tato funkčnost je zatím pouze ve zkušební fázi a do budoucna je třeba ji lépe otestovat a v případě nalezení chyby i odladit. Všechna rozhraní a třídy odpovídají konvencím jazyka Java a vychází z existujících tříd, což zjednodušuje učení tvorby aplikací pro mou platformu.

Při práci na demonstrativních aplikacích jsem si práci s platformou vyzkoušel. Do budoucna bude určitě nutné plně zautomatizovat proces kompilace, ideálně ho plně integrovat do používaného IDE. Jinak je práce s platformou velice jednoduchá a psaní pro ni rychlé.

V rámci dalšího vývoje platformy bude nutné odladit již existující kód do všech detailů a doimplementovat nedokončené části. Platforma se hodí k uveřejnění jako svobodný software, případně jako základ pro konkrétní multimediální projekt. Nabízí se i rozšíření pro jednoduchou práci s trojrozměrnou grafikou a příslušnými formáty.

Tvorba aplikace pro desktop či mobilní zařízení pomocí platformy se tedy nijak neliší. Je třeba pouze brát ohled při použití na mobilních telefonech. Ani v případě rychlosti zde není žádný razantní rozdíl, samozřejmě s ohledem na výkonnost hardwaru.

9 Reference

[1] HEROUT, Pavel. *Java a XML*. 1. vyd. České Budějovice: Kopp, 2007, 313 s. ISBN 978-80-7232-307-4.

[2] SHREINER, Dave. *OpenGL: průvodce programátora*. Vyd. 1. Brno: Computer Press, 2006, 679 s. ISBN 80-251-1275-6.

[3] *Package Index — Android Developers* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/packages.html

[4] *ADT Plugin for Eclipse* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/sdk/eclipse-adt.html

[5] *glDrawElements* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.khronos.org/opengles/sdk/docs/man/xhtml/glDrawElements.xml

[6] *Java3D* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://java3d.java.net/

[7] *JDK 7 Adoption Guide* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://docs.oracle.com/javase/7/docs/webnotes/adoptionGuide/index.html

[8] *java.awt (Java Platform SE 6)* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://docs.oracle.com/javase/6/docs/api/java/awt/package-summary.html

[9] *Overview (Java Platform SE 6)* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://docs.oracle.com/javase/6/docs/api/index.html

[10] *JOGL - Java Binding for the OpenGL API* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://jogamp.org/jogl/www/

[11] *JSON* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://json.org/

[12] *douglascrockford/JSON-java · GitHub* [online]. 2012 [cit. 2012-04-29]. Dostupné z: https://github.com/douglascrockford/JSON-java

[13] *Where Lua Is Used - marbux* [online]. 2012 [cit. 2012-04-29]. Dostupné z: https://sites.google.com/site/marbux/home/where-lua-is-used

[14] *lwjgl.org - Home of the Lightweight Java Game Library* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://lwjgl.org/

[15] *NEWT - JOGL's High Performance Native Windowing Toolkit* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://jogamp.org/jogl/doc/NEWT-Overview.html

[16] *OpenGL ES - The Standard for Embedded Accelerated 3D Graphics* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.khronos.org/opengles/

[17] *OpenGL ES - The Standard for Embedded Accelerated 3D Graphics* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.khronos.org/opengles/sdk/docs/reference_cards/OpenGL-ES-2_0-Reference-card.pdf

[18] *Slovníček pojmů - LGPL - Root.cz* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.root.cz/slovnicek/lgpl/

[19] *YAML: Serializační formát pro ukládání dat* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.zdrojak.cz/clanky/yaml-serializacni-format-pro-ukladani-dat/

[20] *The Official YAML Web Site* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://yaml.org/

[21] *libgdx - Android/HTML5/desktop game development framework* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://code.google.com/p/libgdx/

[22] *Orx - Portable Game Engine* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://orx-project.org/

[23] *Corona: The fastest and easiest way to create cross-platform mobile apps* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.anscamobile.com/

[24] *Unigine: real-time 3D engine (game, simulation, visualization and VR)* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://unigine.com/

[25] *UNITY: Unity 3 Engine Features* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://unity3d.com/unity/engine/

[26] *Open Source Initiative OSI - The BSD 3-Clause License* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.opensource.org/licenses/BSD-3-Clause

[27] *GNU General Public License, version 3 (GPL-3.0)* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.opensource.org/licenses/GPL-3.0

[28] *The GNU Lesser General Public License, version 3.0 (LGPL-3.0)* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.opensource.org/licenses/LGPL-3.0

[29] *Open Source Initiative OSI - The MIT License (MIT):Licensing* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://www.opensource.org/licenses/MIT

Přílohy

Přiložené CD obsahuje zdrojové kódy i spustitelné ukázkové příklady ve formě kompilátů. Ke spuštění doporučuji Windows Vista nebo vyšší (ukázky jsou připraveny pro obě bitové verze). Je možné i spuštění na operačním systému Linux, ale pro něj nejsou připraveny spouštěcí dávky.

Pro správné spuštění příkladů pro Android je nutné mít správně nainstalováno a nastaveno Android SDK. V systémové proměnné *Path* musí být přidána (středníkem oddělena) cesta ke složce **android-sdk/platform-tools**, aby systém nalezl příkaz **adb**. Taktéž je nutné mít spuštěno virtuální zařízení s připojenou SD kartou. Spouštěcí dávky upravují práva souborů, nedoporučuji tedy zkoušet ukázky na reálném zařízení.

Obsah CD

- **android/** - obsahuje mobilní část implementace a ukázkové příklady pro ni, současně jde o projekt pro prostředí Eclipse s ADT pluginem
  - **src/** - obsahuje zdrojové kódy
  - **demo *?* .bat** - spuštění jednotlivých ukázek
- **desktop/** - obsahuje desktopovou část implementace a ukázkové příklady pro ni, současně jde o projekt pro prostředí Eclipse
  - **doc/** - obsahuje vygenerovanou dokumentaci
  - **lib/** - obsahuje nativní knihovny pro JOGL
  - **src/** - obsahuje zdrojové kódy
  - **demo *?* .bat** - spuštění jednotlivých ukázek
  - **demo *?* _x64.bat** - obdoba pro 64-bitové systémy
- **tex/** - zdrojové kódy této práce včetně obrázků
- **Bakalářská práce.pdf** - samotná práce v elektronické podobě